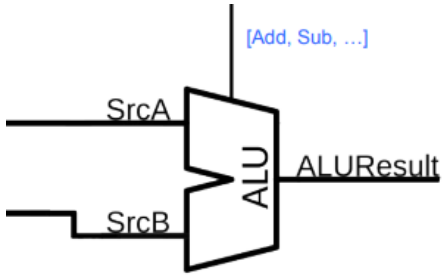
Writeup: Sign Extension beim addi Befehl

Wofür braucht man eine Sign extension?

Da alle Instruktionen 32 Bit groß sind, können wir nicht beliebig große Immediates (Konstanten) den Instruktionen mitgeben. Wie viele Bits für den Immediate in einer Instruktion reserviert ist, hängt natürlich von der spezifizierten Instruktion ab. Beim addi Befehl hat man nach Abzug des Opcodes und weiterer Steuerbits genau 12 Bit übrig, in welcher ein Immediate kodiert werden kann.

Kurzer Exkurs in die Prozessorarchitektur (Details folgen später in der LV): Eine der wichtigsten Komponenten eines jeden Prozessors ist die ALU (Arithmetic Logic Unit), oft Rechenwerk genannt. Sie ist verantwortlich für das Berechnen von Ergebnissen aus mathematischen/logischen Rechenoperationen (Addition, Subtraktion, ..., AND, OR, ...). Dazu erhält sie 2 Operanden, jeweils 32 Bit groß und liefert wiederum ein 32 Bit großes Ergebnis.

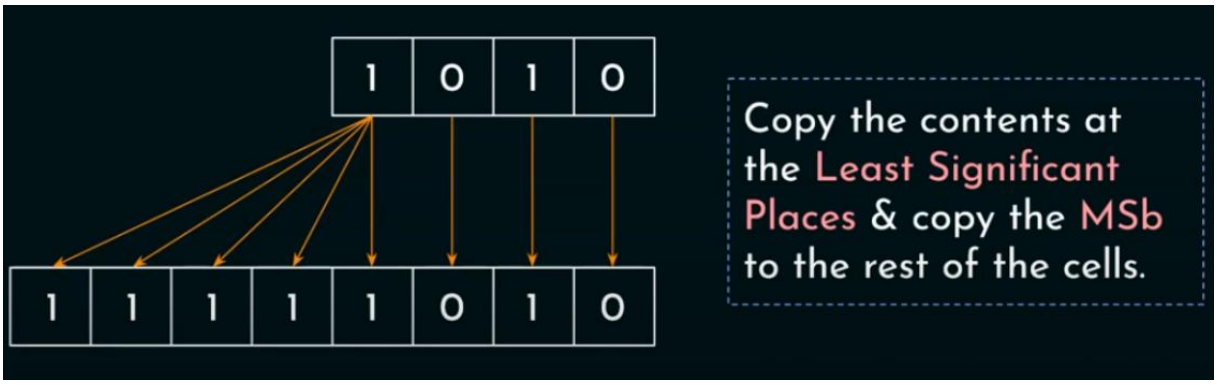


Warum sind die Eingänge 32 Bit groß? Ganz grob: die ALU soll auch mit Registern rechnen können (z.B. diese Addieren in der ‘add‘-Instruktion) und diese Register sind im rv32 bekanntlich 32 Bit groß.
Zurück zum addi-Befehl, welcher folgendermaßen strukturiert ist:
addi zielregister, quellregister, immediate
Bei der Dekodierung im Prozessor wird da das Quellregister am SrcA und der Immediate an SrcB angelegt. ↯ Unser 12 Bit Immediate passt offensichtlich nicht zu den erwarteten 32 Bit.
Was nun? Wir müssen die 12 Bit auf 32 Bit erweitern. Bei der konkreten Umsetzung gibt es folgende 2 Möglichkeiten:

1. Die restlichen 20 Bit mit 0 auffüllen.
   Beispiel positive Zahl: (12 Bit): $(15)_{10}$ = 0xF = 0b1111
   → (Erweitert auf 32 Bit): $(15)_{10}$ = 0xF = 0b1111 ✅
   Beispiel negative Zahl: (12 Bit): $(-534)_{10}$ = 0xDEA = 0b1101.1110.1010
   → (Erweitert auf 32 Bit): 0x000.0DEA =
   0b0000.0000.0000.0000.0000.1101.1110.1010 = $(3562)_{10}$ ↯ Durch das erweitern durch 0en wurde unser negativer Immediate positiv (MSB=0)!

2. Sign extension: das (linkeste Bit) Most Significant Bit (MSB) wird angeschaut: Falls MSB=1, werden die 20bit mit 1en aufgefüllt. Falls MSB=0, werden die 20bit mit 0en aufgefüllt.
   Beispiel positive Zahl: (12 Bit): $(15)_{10}$ = 0xF = 0b1111
   → (Erweitert auf 32 Bit): $(15)_{10}$ = 0xF = 0b1111 ✅
   Beispiel negative Zahl: (12 Bit): $(-534)_{10}$ = 0xDEA = 0b1101.1110.1010
   → (Erweitert auf 32 Bit): 0xFFFF.FDEA = 0b1111.1111.1111.1111.1111.1101.1110.1010
   Das MSB ist eine 1, also ist unsere negative Zahl schonmal negativ geblieben ✅.
   Wir berechnen das Zweierkomplement, um den Betrag der negativen Zahl zu bestimmen:
   0b1111.1111.1111.1111.1111.1101.1110.1010
   1-Komplement: 0b0000.0000.0000.0000.0000.0010.0001.0101
   2-Komplement: 0b0000.0000.0000.0000.0000.0010.0001.0110 = $(-534)_{10}$ ✅

Unter anderem, weil wir in RISC-V negative Immediates verwenden, um zu subtrahieren (wir verwenden addi a0, a0, -(Immediate) da kein subi Befehl existiert) hat man sich in RISC-V dazu entschieden die Sign-Extension umzusetzen (**Designentscheidung**).



Mit dem neuen Wissen über die Sign Extension können wir nun die Tutoraufgabe sinnvoll lösen.

Tutoraufgabe 2 b)

Laden Sie die Konstante 0xABAD1DEA in das Register a3. Verwenden Sie keine Pseudobefehle.

0xABAD1DEA besteht aus 8 Hex-Zeichen, ein Hex-Zeichen entspricht 4 Bit, also haben wir insgesamt eine 32 Bit Zahl. Wir wissen also, dass die Zahl nicht durch ein einfaches addi ins Register geladen werden kann, da wir addi nur 12 Bit mitgeben können.

Wir laden also zunächst die oberen 20 Bit in das Register und addieren darauf die letzten 12 bit unsere Zahl:

lui a3, 0xABAD1

Register a3 sieht jetzt folgendermaßen aus: 0xABAD1000

addi a3, a3, 0xDEA

Da wir nun wissen, dass addi den Immediate sign extended müssen wir uns ansehen, ob das MSB = 1 ist. 0xDEA = 0b1101.1110.1010 = $(-534)_{10}$. Wir sehen, dass das MSB = 1 ist, also wird unser Immediate auf 32 Bit mit 1en aufgefüllt und zum Registerinhalt addiert

0xDEA → (Sign Extension) → 0xFFFF.FDEA

0xABAD1000 + 0xFFFF.FDEA = 0xABAD0DEA ↯ Durch die Sign-Extension erhalten wir nicht mehr das erwartete Ergebnis.

Neuer Lösungansatz: Den lui-Immediate mit dem sign extendeden addi-Immediate neu berechnen.

Wir benötigen ein y, sodass gilt:

0xABAD1DEA = y + 0xFFF.FDEA

y = 0xABAD1DEA - 0xFFF.FDEA

y = 0xABAD1DEA – (- 0x0000.0216)

y = 0xABAD1DEA + 0x0000.0216

y = 0xABAD2000

Nebenrechnung 2-Komplement 0xFFFF.FDEA.
1-Komp: Alle F werden 0, wir betrachten nur 0xDEA = 0b1101.1110.1010
→ 0b0010.0001.0101

2-Komp: 0b0010.0001.0110 = 0x216

Also laden wir y in die oberen 20 Bit und addieren den sign-extendeten Immediate darauf.

lui a3, 0xABAD2000

addi a3, a3, -534

0xABAD2000 + 0xFFFF.FDEA = 0xABAD1DEA ✅

Kleines Detail am Rande: Wertebereiche

Warum haben wir im letzten Schritt addi a3, a3, -534 geschrieben und nicht addi a3, a3, 0xDEA? Probiere mal letzteres in qtrvsim aus und du wirst eine Fehlermeldung dafür bekommen, zurecht!

Wir wissen, dass in die addi-Instruction ein Immediate von 12 bit kodiert wird. Da wir im Zweierkomplement arbeiten ergibt sich daraus der Wertebereich (siehe Tutorium letzte Woche): -2^{12-1} bis $2^{12-1} - 1$

0xDEA = 0b1101.1110.1010

2^{11} = 1000.0000.0000

$2^{11} - 1$ ist demnach = 0111.1111.1111 (unser größter abspeichbarer Wert)

An den Binärzahlen sieht man offensichtlich, dass sich das 0xDEA außerhalb unseres Zahlenbereichs befindet.

Wenn wir nun addi a3, a3, -534 (Analog mit addi, a3, a3, -0x216 da 0xDEA = $(-534)_{10}$ = 0x216) anwenden, wird der Betrag genommen und mithilfe des Zweierkomplements (11 Bit)korrekt innerhalb des Wertebereichs abgespeichert

$(-534)_{10}$ = 0x216 = 0b010.0001.0110

1-Komplement = 0b101.1110.1001

2-Komplement = 0b101.1110.1010 = (0x5EA)

Durch die anschließend erfolgende Sign-Extension erhält man wieder 0b1111.1111.1111.1111.1111.1101.1110.1010 = (0xFFFF.FDEA)

Der Betrag der negativen Zahl also $(534)_{10}$ bzw. 0x216 bleibt dabei immer erhalten. (Wegen der Sign Extension) ✅